taste

A real-time software engineering tool-chain
Overview, status and future

Maxime Perrotin
Eric Conquet
Julien Delange
Thanassis Tsiodras
André Schiele

European Space Agency

www.esa.int

what is taste?

- A software engineering tool-chain targeting heterogeneous systems
- A set of tools to build and validate embedded, real-time systems
- A concentrated set of technology, available for free and open source

taste

The Assert Set of Tools for Engineering

heterogeneous systems (1)

fault detection

mode management

navigation

resource management

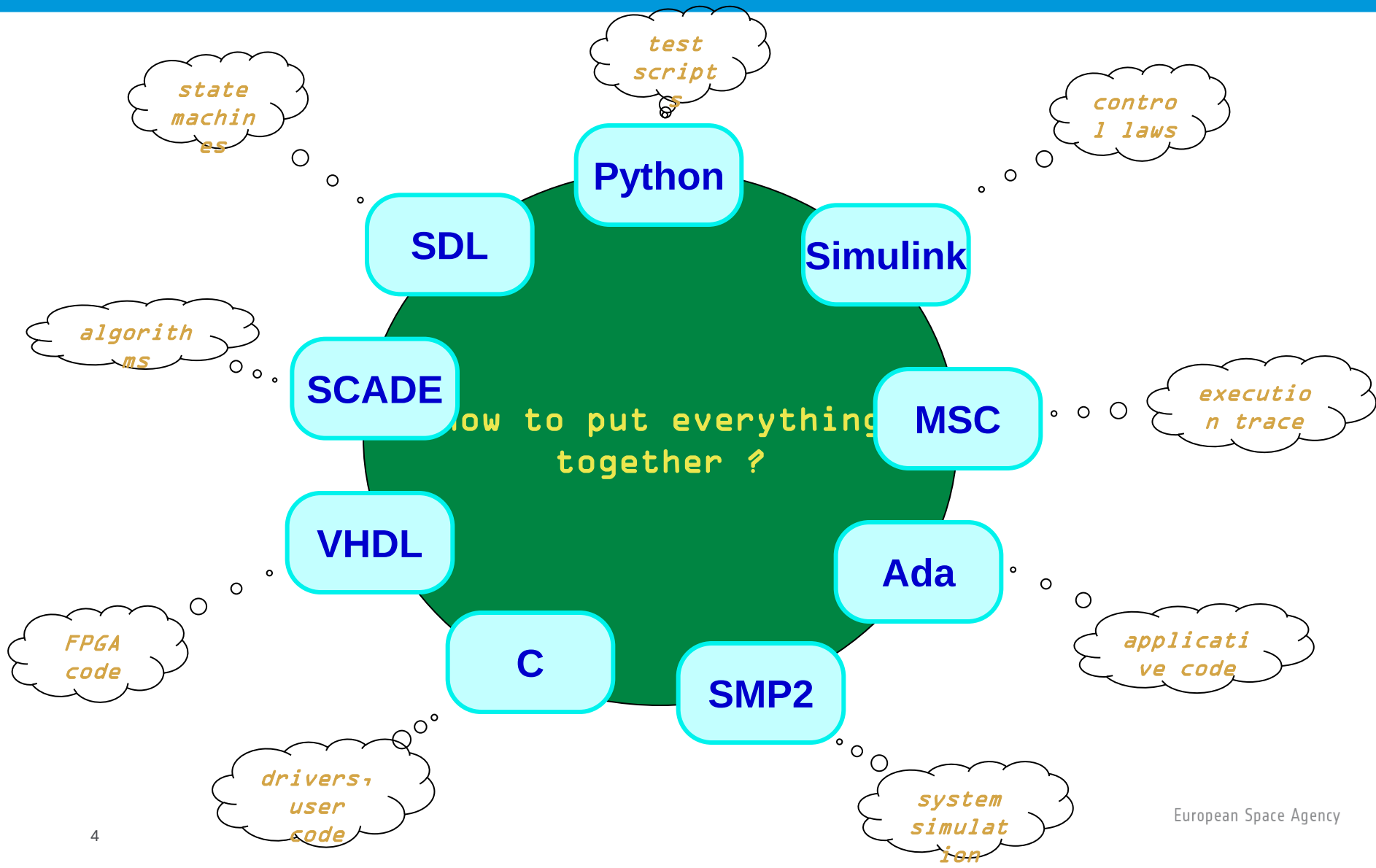
drivers

communication

companies
X, Y, Z…

companies
A, B, C

heterogeneous systems needs (2)



how to put everything together?

- **Manually?**
  - Requires a lot of hacking
  - Difficult maintenance in case of interface changes
  - That is the most common way of doing

- **Using a commercial modelling tool?**
  - No support for heterogeneous models (at best, Simulink integration)
  - No support for sensor/actuators interfacing

- **Using TASTE**

example

hardware interface (1)

Communication & State Management (SDL/RTDS)

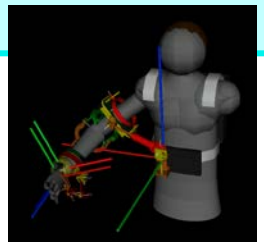
hardware interface (2)

Exoskeleton (Sensors)

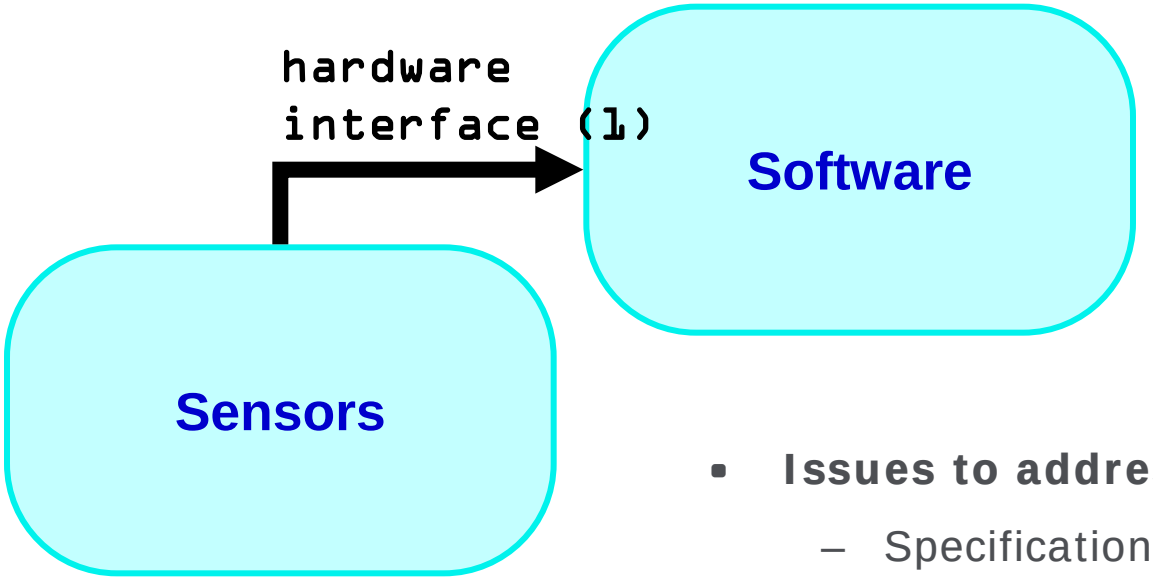
software interface

Control law (Simulink)

Robotic arm (Actuators)

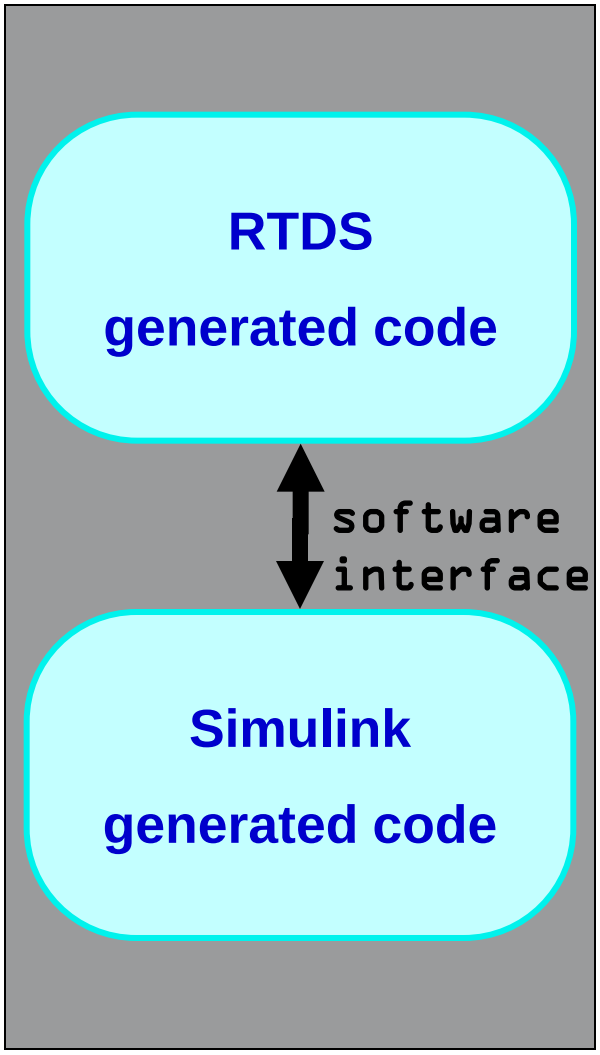


interface issues



- **Issues to address**:
  - Specification of the interface (logical message description)
  - Message physical representation (binary stream) – imposed by the hardware. Conversion to a software data structure

interface issues (2)



- **Issues to address**:
  1) Specification of the interface (logical message description)
  2) Conversion at model level (keep the same semantics in both RTDS and Simulink)
  3) Conversion at code level: map each field of the interface from one generated piece of code to the other

taste solves these issues

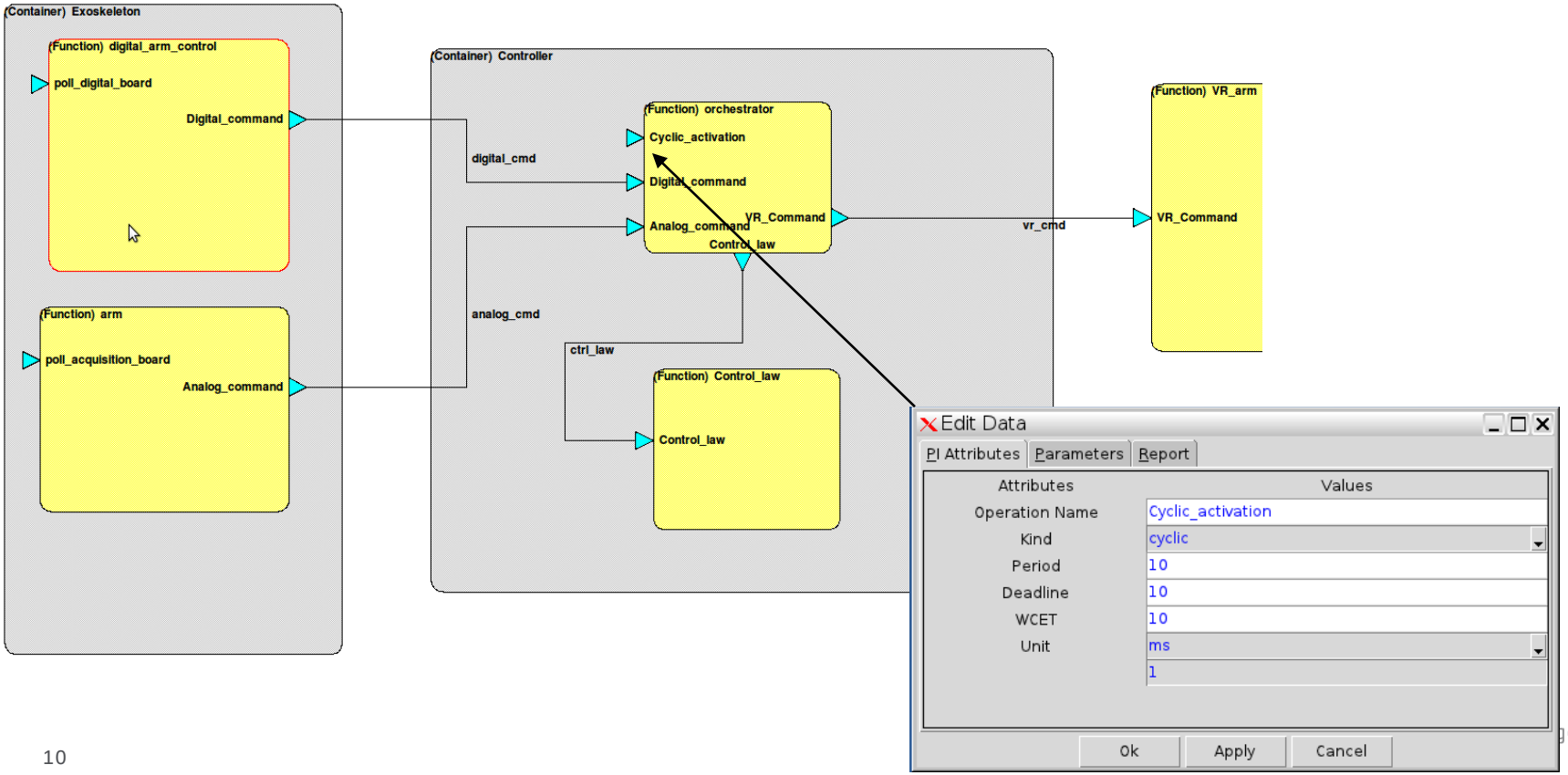
- TASTE solves integration issues by allowing all hardware and software pieces to communicate together transparently
- TASTE relies on two key formally-defined languages:

ASN.1 **AADL**

- TASTE relies on a clear and sound process
- TASTE is designed to be used by non-software specialists
- TASTE does not reinvent the wheel: it uses existing, proven technologies (SDL, MSC, SCADE…) and put them together in order to build and validate homogeneous systems

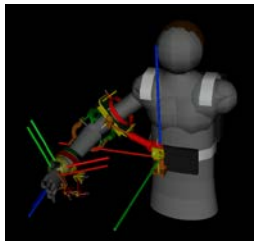
AADL

- A textual notation to capture all the attributes of a system
- TASTE provides a graphical view of AADL files



ASN.1 to describe interfaces

- A simple notation to describe software and hardware interfaces
- Our tools generate code for embedded systems (no malloc, no system call, support for C and [Spark] Ada)



Edit and load Data View

dataview.asn

```
-- Output types

VR-Model-Output ::= SEQUENCE {
    x1 REAL (-1000 .. 1000),
    y1 REAL(-1000 .. 1000),
    z1 REAL(-1000 .. 1000),
    p1 REAL(-1000 .. 1000),
    x2 REAL(-1000 .. 1000),
    y2 REAL(-1000 .. 1000),
    z2 REAL(-1000 .. 1000),
    p2 REAL(-1000 .. 1000),
    x3 REAL(-1000 .. 1000),
    y3 REAL(-1000 .. 1000),
    z3 REAL(-1000 .. 1000),
    p3 REAL(-1000 .. 1000),
    j-rad SEQUENCE (SIZE(16)) OF REAL (-1000 .. 1000)
}
```

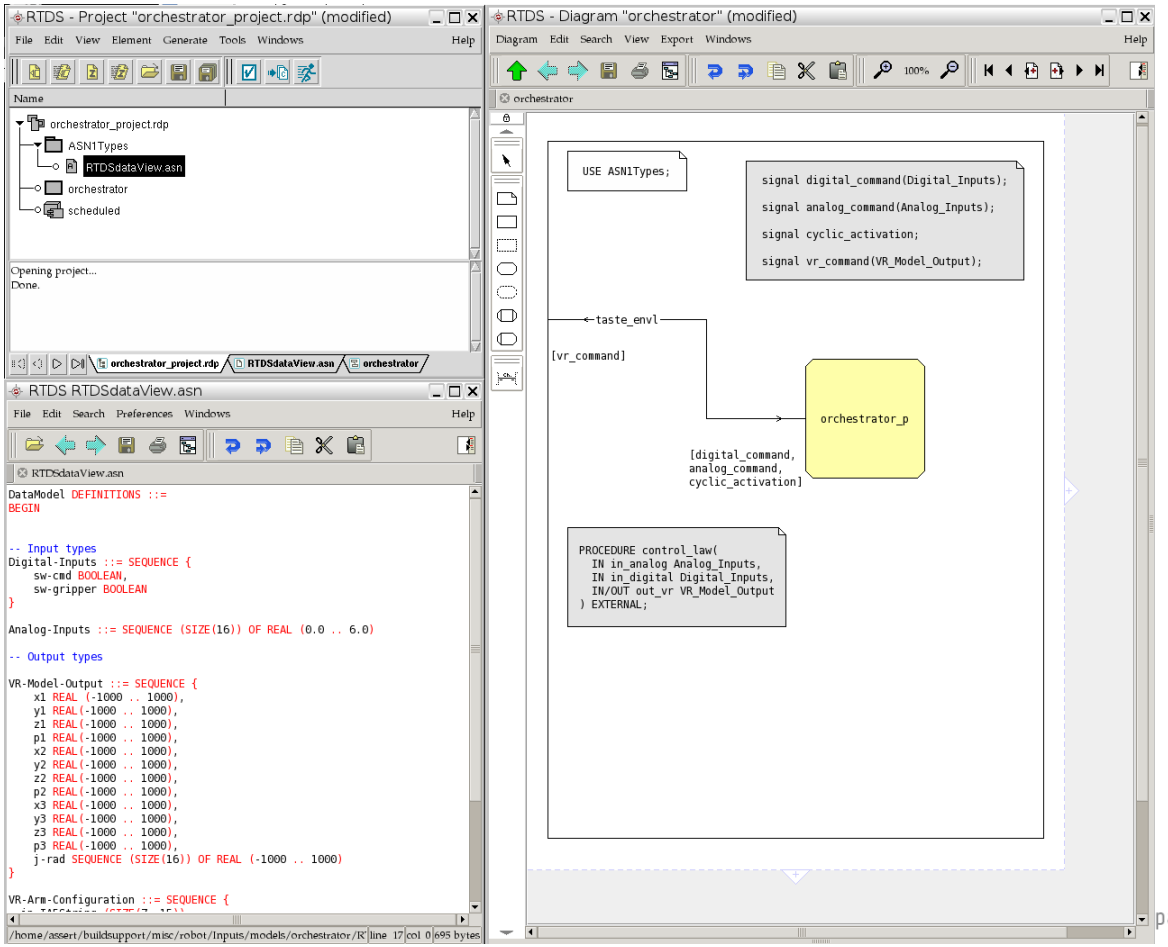
dataview-uniq.acn

```
/*Output types*/
VR-Model-Output []{
        j-rad [size 16] {
                dummy [encoding IEEE754-1985-64, endianness little]
        },
        p1 [encoding IEEE754-1985-64, endianness little] ,
        p2 [encoding IEEE754-1985-64, endianness little] ,
        p3 [encoding IEEE754-1985-64, endianness little] ,
        x1 [encoding IEEE754-1985-64, endianness little] ,
        x2 [encoding IEEE754-1985-64, endianness little] ,
        x3 [encoding IEEE754-1985-64, endianness little] ,
        y1 [encoding IEEE754-1985-64, endianness little] ,
        y2 [encoding IEEE754-1985-64, endianness little] ,
        y3 [encoding IEEE754-1985-64, endianness little] ,
        z1 [encoding IEEE754-1985-64, endianness little] ,
        z2 [encoding IEEE754-1985-64, endianness little] ,
        z3 [encoding IEEE754-1985-64, endianness little]
}
```

C Tab Width: 8 Ln 1, Col 1 INS

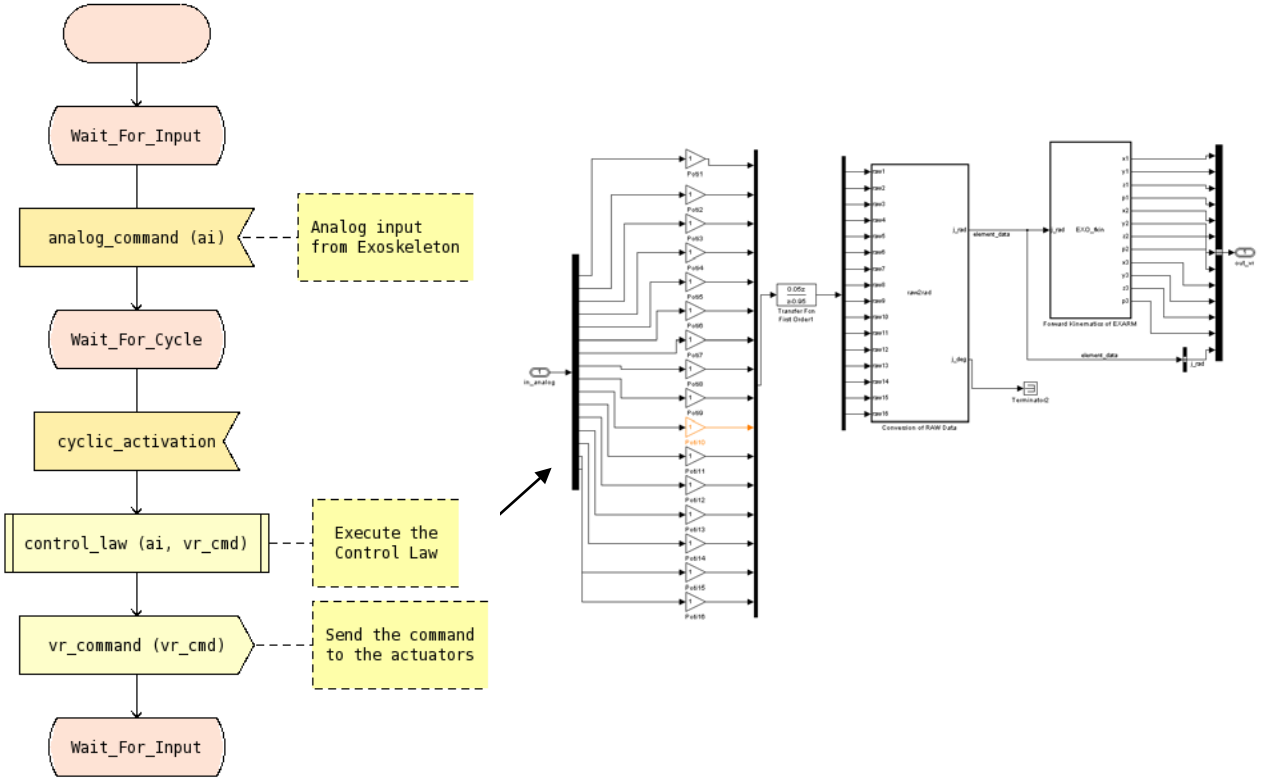
start the real work from function skeletons

- Supported languages: SDL (RTDS, ObjectGEODE), SCADE, Simulink, C, Ada, VHDL



Mix languages to get the best of all worlds – no “unified language” to rule them all!

- The robotic case study mixes C (drivers), SDL (RTDS – system overal orchestration and logic) and Simulink (control laws)



A straightforward process

1) **Describe interfaces with ASN.1**

A straightforward process

1) Describe interfaces with ASN.1

2) Capture the logical architecture using the AADL editor *(Interface View)*

A straightforward process

1) Describe interfaces with ASN.1

2) Capture the logical architecture using the AADL editor *(Interface View)*

3) Generate code skeletons and write the deployment code

A straightforward process

1) Describe interfaces with ASN.1

2) Capture the logical architecture using the AADL editor *(Interface View)*

3) Generate code skeletons and write the deployment code

4) Capture the system hardware and deployment *(Deployment View)*

A straightforward process

1) Describe interfaces with ASN.1

2) Capture the logical architecture using the AADL editor *(Interface View)*

3) Generate code skeletons and write the deployment code

4) Capture the system hardware and deployment *(Deployment View)*

5) Verify system feasibility using TASTE-provided tools (Cheddar, MAST, REAL)

A straightforward process

1) Describe interfaces with ASN.1

2) Capture the logical architecture using the AADL editor *(Interface View)*

3) Generate code skeletons and write the deployment code

4) Capture the system hardware and deployment *(Deployment View)*

5) Verify system feasibility using TASTE-provided tools (Cheddar, MAST, REAL)

6) Build the system and download it on target

A straightforward process

1) Describe interfaces with ASN.1

2) Capture the logical architecture using the AADL editor *(Interface View)*

3) Generate code skeletons and write the deployment code

4) Capture the system hardware and deployment *(Deployment View)*

5) Verify system feasibility using TASTE-provided tools (Cheddar, MAST, REAL)

6) Build the system and download it on target

7) Analyze system performance at run-time

A straightforward process

1) Describe interfaces with ASN.1

2) Capture the logical architecture using the AADL editor *(Interface View)*

3) Generate code skeletons and write the deployment code

4) Capture the system hardware and deployment *(Deployment View)*

5) Verify system feasibility using TASTE-provided tools (Cheddar, MAST, REAL)

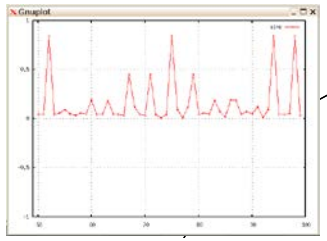
6) Build the system and download it on target

7) Analyze system performance at run-time

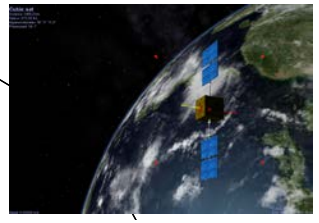
8) Validate implementation regarding specifications

taste integrated runtime features

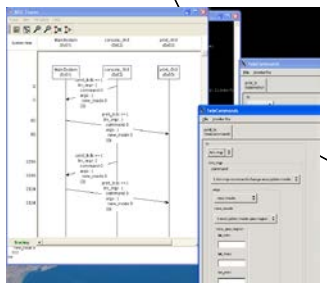
- Auto-GUI generation
- MSC tracing and recording
- Plot streaming
- Testing via python scripts
- Code coverage
- Profiling
- Task analysis
- 3D rendering



GNUPlot streaming



Celestia rendering



GUI and MSC control

```
from DataView_asn import *

def testReal(val):
    # Create a stream buffer to host the encoded data
    d1 = DataStream(DV.T_REAL_REQUIRED_BYTES_FOR_ENCODING)
    # Create a T_REAL
    f1 = T_REAL()
    # Set the value
    f1.Set(val)
    try:
        # Encode the value of f1 into the buffer d1
        f1.Encode(d1)
    except:
        print "Encoding failed..."
        sys.exit(1)

    binaryData = d1.GetPyString() # Get the encoded stream bytes
                                  # as a Python string. You can
                                  # pass these data over sockets,
                                  # save them to files, etc
```

Python test scripts

TASTE binary

before the demonstration

- In addition to ESA, TASTE main contributors are
  - Semantix (GR)
  - Ellidiss (F)
  - ISAE (F), ENST (F)
  - UPM (ES)

- TASTE is available freely and open source
  - Ensure long-term support

- It can be downloaded from:
  - **www.assert-project.net/taste**


esa
For more information:
www.assert-project.net/taste
European Space Agency